Address Cleanup Business Rules for the Geocoding

Restaurant Meals Program Report

Background

We received a request from our Restaurant Meals program to produce a map and reports that determine the total number of program cases within (and restaurants servicing) each: zip code, city, unincorporated community, and supervisorial district.

We were given a table with 546,000 plus case records. Of these records, over 50,000 were not geocoding. Almost 20,000 were homeless cases, with no address.

The remaining 30,000 plus records seemed to have had issues with how they were entered into the system. For these, we created a cleanup script that reduced the number of un-geocoded records to within 3,200 records.

Patterns Identified

One of the things that jumped off the screen to us when reviewing the list of records that failed to geocode, was the pattern of how they were stored in the table. They had issues such as fractions within the number portion of the address (such as ½, ¼, etc.). Several hundred had issues with lack of spacing, where the address is all bunched together (1234MainStreet). Other common patterns included a “th” at the end of a street name, where an “st” was needed (“21th” as opposed to “21st”); a hyphen or letter in the number portion of the street address (349-B, a common one in the AV area of the County); and problems with misspelled street names (“Standford” vs “Stanford”) or city name such as (“LosAngeles” vs “'Los Angeles”).

A quick point about the city field, it was not surprising that Los Angeles was the only regularly misspelled/erroneously entered city given the population density and high probability most cases would be there.

Cleanup Script Applied

We commenced to create a cleanup script to correct these and other similar issues in the primary programming language we had at our disposal (in regards to knowledge) at the time, ColdFusion. Depending on user preference and availability of resources and technology, any programming language could be used to do the same thing.

Assuming a copy of the database has already been created, here are the basic steps we went through with the script:

- For each record;
    - Set a temporary variable (appropriately scoped for your programming environment) for each address field.
        - A temporary variable was used for the street and city in our case. We could have also created a temporary variable for zip code however; we found it less necessary as zip codes are less likely to be incorrect.

    - Group the targeted changes, such as the street fractions mentioned above. We chose fractions first because these were the most common type of hang up to geocoding an address. Because there are dozens of possible fraction combination, (and because we have since learned a little more since we last coded this script), a regular expression (RegEx) pattern recognition code can and should be used in your program script (see attached sample code). I did a recent test and not only did it improve performance slightly, it trimmed down the amount of code needed, and made it easier to maintain.

Adding to or extending the pattern as needed makes the most sense, once a good RegEx pattern has been established. Here is a sample of the type of Regular Expression we used to find fractions:

- If myVariable contains [0-9]/[0-9], then set this part of the variable [0-9]/[0-9], to " " (set it to a single space);
  - [0-9] means any number from 0 to 9. The "/" is matched explicitly, within the pattern of any digit from 0 to 9 on either side of it. We can easily extend the pattern to match multiple digits by adding and additional bracket sequence. So for example if by chance there was a fraction with 10 units (I did not see any in my case), we can simply extent the pattern to "[0-9] [0-9]/[0-9] [0-9]" and now we have modified the pattern to match any potential double digit fractions (though this is exceedingly unlikely). The most common fraction I found was ½.

- Next we targeted some commonly misspelled street names.
  - Hyphens were also common within addresses and these failed to geocode. Simply removing these hyphens proved helpful. One other gotcha we found regarding hyphens was their use in the first part of the address. This along with the use of letters in the number part of the street address caused geocoding issues.
  - Finally, we used a method to replace any additional spaces in between the street address.
- It proved useful to output the results visually, before committing changes to back into the database. We displayed the results in the form of a test output on a webpage, highlighting changes in red.
- Once we were satisfied with the types of changes we saw, we updated the database with their new values.

Summary

In summary, there were lessons learned in our address cleanup process. One thing I would have liked to use more of, were regular expressions to match patterns instead of just picking common cases in the data. That said, a visual review and identification of the pattern of problems in the addresses data is invaluable, as different systems will have their own issues with how address data is entered into that system (one can't replace getting to know our data intimately). And , regular expression patterns need to be tailored to each situation.

These lessons learned should help our Department create better address capture processes at the data entry point of new systems. In our case, we were dealing with a legacy system that will cost millions to replace and updates to that system at this stage are highly unlikely (a new system is in the works).